SysCon 2008 – IEEE International Systems Conference
Montreal, Canada, April 7–10, 2008

Pervasive Computing: Migrating to Mobile Devices: A Case Study

Shihong Huang[1], Jan Mangs[2]
[1]Department of Computer Science & Engineering, Florida Atlantic University
777 Glades Road Boca Raton, FL 33431
Phone: +1-561-297-1275, Email: shihong@cse.fau.edu
[2]Department of Computer Science & Engineering, Florida Atlantic University
777 Glades Road Boca Raton, FL 33431
Phone: +1-561-306-4376, Email: jmangs@fau.edu

Abstract *– The concept of pervasive computing is that computers and their applications will seamlessly integrate into our daily lives. There are many areas of research in pervasive computing such as context-awareness, device and network heterogeneity issues, power consumption, and connectivity. Each area intends to address problems such as how devices interpret their environment, and how devices communicate with each other. While there is much potential for developers to create pervasive applications from scratch, it is much more likely that they are able to reuse existing applications and documentation for mobile devices. This paper addresses the specific problems of porting pervasive applications on mobile devices in general as well as the unique problems presented by mobile devices which restrict implementation methods. It proposes a framework based on model driven engineering concepts for developing software for restrictive pervasive systems. The paper also considers the implications of the preliminary framework on the iPhone, which forces developers to work through its Safari browser.*

Keywords *– pervasive applications, restricted mobile devices, software migration, portability, iPhone*

I. INTRODUCTION

As mobile devices capabilities have continued to improve, the interest in pervasive or ubiquitous computing has grown steadily. Nowadays cell phones can be found nearly everywhere and offer multiple features aside from just making phone calls. The idea that a small device could run an application that dynamically adapts to environmental conditions would have seemed like science fiction twenty years ago now have become reality.

Pervasive computing is a result of the growth of mobile devices that provide more processing power and memory in a ever-shrinking devices. Pervasive or ubiquitous computing is the idea that computers will seamlessly integrate into humanity's world to the point that no user input is required for a computer to understand what it should do. For example, when a homeowner walks into his house late at night, the computer system running behind the scenes in his house should sense that its owner is home and turn on the lights and perhaps raise the temperature to his owner's normal preference. When the owner goes to sleep, the system should notice that no movement is occurring and its inhabitants are sleeping. Thus it will dim or turn off all the lights in the house and activate the security alarm. Although there is no system in existence that perform all these functions, the aim of research in pervasive computing is to one day bring this type of functionality and self-awareness to a computer system.

The key problem in developing pervasive applications is device interoperability. Different mobile devices often provide different functionality and have different CPU speeds, memory capacity, and power. This means that an application created for one platform may or may not run on a different device. One device may have a GPS and Wi-Fi connection while another device has neither. Software also plays a part in differences between devices. One device such as Nokia's N95 may provide developer support for Java and C++ while another device such as Apple's iPhone only supports JavaScript and AJAX for developers. Just as devices are different, so are the networks which support them. This means two devices may not be able to communicate with each other with a type of middle broker that connects the two.

As phones gain improved features the difficulty in developing software increases. Older software may have trouble running on newer phones and legacy systems often are not able to run at all on mobile devices. For example, a banking system written in COBOL before the advent of the Internet may have limitations communicating devices that exist outside of their immediate location. In [1], Yang et al have worked on a middleware that allows mobile devices to work with existing legacy systems by creating a type of wrapper agent around the legacy system. Their approach leverages both the existing software and creates an application that does not tax the mobile clients too heavily.

Another middleware called Mobile Adaptive Data Accessing Middleware uses a similar method [3]. The proposed architecture of MADAM uses an integrated semi-SOA approach in which both performance and extensibility are leveraged. The client's side uses a

context structure that is based on characteristics of the client's device and the server side assigns specific executants to an end server. The early prototype which was created ran three different clients on three different devices, all with the same functionality. These types of middleware schemes are common when dealing pervasive applications because of the lack of resources on the mobile devices software runs on. It allows the device to use its own mobility to its advantage while having a server do the heavy duty processing and computation.

Other middleware frameworks attempt to address other issues such in mobility and context-awareness include architectures such as LIME [8], and XMIDDLE [9] using XML and peer to peer constructs. There have also been proposals for programmable event-based architectures for mobile environments [2] [10].

In another scenario, developers may wish to port an application to mobile devices from an existing desktop application. In this case, the application has already been designed to run primarily in an environment in which the user does not migrate from place to place and has a developed, mature interface.

II. PERVASIVE APPLICATION DEVELOPMENTS AND MOBILE DEVICES

The unique aspects of pervasive applications require a new approach in the normal software development paradigm. A certain approach may work for one set of mobile devices only to be less effective on another set. Software engineering paradigms are applicable to pervasive computing as long as the additional constraints are taken into consideration.

In this section, we take into consideration the work done in the area of pervasive application development. Previous experience developing generalized frameworks for application development can be used to evaluate how effectively we can port software for 'software-restricted' devices. This section discusses some of the concepts that can possibly be used in application development for restricted mobile devices, such as the iPhone.

A. Early / Rapid Prototyping

Prototyping is a method commonly used in software engineering to provide a proof-of-concept model or test whether a given design will work as intended without sacrificing too much time and cost to the effort. Because of the inherent complexity in pervasive systems, prototyping should and will play an important role in any new software development. Effectively evaluating the correctness of a prototype is a challenge to any developer and is even more so when dealing with pervasive systems.

In an article titled "Evaluating Early Prototypes in Context: Trade-offs, Challenges, and Successes," the authors relate their experience with two methods of prototyping known as experience prototyping and Wizard of Oz prototyping [5]. The first method is a variant of experience prototyping in which researchers and developers themselves become participants in actual scenarios. The other method, Wizard of Oz prototyping, involves creating mock functionality in some of the prototype to create the illusion of having a functioning system. In return for a more complete prototype, some accuracy and granularity is lost. The findings in the article are somewhat expected; experience prototyping is more realistic but less impartial while Wizard of Oz prototyping is more impartial but less realistic. As long as a researcher knows of the tradeoffs and benefits of each approach, they can be used to as an alternative to more traditional prototyping.

Although early prototyping is not used in the preliminary framework presented in this paper, the concepts behind it could provide a benefit to the preliminary framework. Because of the additional software constraints on the iPhone and other restricted devices, prototyping allows developers to determine if an implementation is feasible.

Notably, two of the creators of the tools described in the next section have also done work on prototyping for pervasive applications. Weis and Knoll [6] have contributed to a tool called Visual RDK that allows developers to create prototypes for pervasive applications visually.

B. Automatic Configuration

One software development process to consider is a combination of component abstraction system called PCOM [13] and a graphical programming language called Nexel [4]. The approach is based on the same tried and true process of software development for desktop software. PCOM and Nexel form a tool chain. First, a developer creates various application components trees where one component provides functionality to its parent by relying the functionality of its children. To accomplish this, contracts are assigned to components that declare its dependencies. When a component is installed on a device, it is automatically configured to run without user input. Once a component tree has been created in PCOM, it can be customized and viewed graphically using Nexel.

While Nexel is a useful addition, the main point we concentrate on is PCOM. Typically, automatic configuration is attractive because of the lack of user

configuration. It is even more important for the type of restricted devices where the user has limited ability to set up manually his device such as the case in this paper. Although it may not be possible to utilize PCOM on all restricted devices such as the iPhone at the moment, the ideas behind it are useful in development environments with special restrictions. Depending on how severe the restrictions are on a specific device, some may not be able to utilize automatic user configuration and the viability of PCOM is limited. In our mini case study, the iPhone does not allow for automatic user configuration but it's important to reiterate that the ideas presented should be heeded when dealing with pervasive programming and development in general.

C. Model Driven Development

Model driven development, simply put, is a software engineering technique that aims to generate code from models automatically. MDD has already shown some potential to become a powerful technique for creating software for various areas. The underlying concept behind MDD is that developers will eventually only need one model (Platform Independent Model) to generate code on any platform by using a serious of transformations [17]. That idea alone is incredibly powerful in pervasive computing considering that device heterogeneity continues to be a problem. If it were possible to automatically generate code from a set of models for all of the devices a developer has, pervasive application development would become much more streamlined.

Pham et. al. [7] discuss the available MDD frameworks for pervasive systems and the benefits of applying MDD to pervasive systems. Specifically, they talk about their work on applying MDD to the development of a Search and Rescue support system called Canine Augmentation Technology [12]. Its intent was to supplement USAR canine teams engaged in Search and Rescue where animals are used to search unsafe or remote areas. The system included components for the dog such as GPS, video, and audio, a system for the dog handler, and an observer system which monitors the area undergoing a Search and Rescue.

Because of the mobile nature of the system, it resembles a pervasive system. Pham et al. noticed the similarities between the different SAR systems and decided to create graphic models for each component. From there, they followed the MDD methodology to reuse software components of the original CAT system. Although it can be debated as to how close to a pervasive system CAT is, the similarities are great enough to validate their argument that MDD is applicable to pervasive systems.

The ideas behind model driven development will be a key driving point to the framework proposed in this paper. Because pervasive applications need to run on multiple platforms, model driven development paradigms such as MDA [11] provides an effective solution with platform independent models and platform specific models. Transformations between different models and source code may not be perfect but can still provide a rudimentary enough functionality to assist in porting between systems. Although pervasive systems are more complex than more traditional development environments, they should still be able to be modeled effectively by developers with sufficient experience and the right tools. MDD also promotes systematic reuse and allows for easier simulation and prototyping with models, which means that it would work well in conjunction with early prototyping methods to determine the effectiveness of an idea.

III. A PRELIMINARY FRAMEWORK FOR PORTABILITY EVALUATION

The framework proposed by this paper is one which aims to provide the most efficient method of porting pervasive software to a mobile device. It hopes to provide the simplest transition possible without a great deal of direct, repeated human intervention. The skeleton structure and some basic ideas are presented in this section and will be continued to be built upon.

A. Evaluating the Source Application

The initial step shall be to determine portability of a source application that a developer wishes to port. Some applications may not be suitable or possible to port to another platform because of large size, high calculation requirements, low quality source code, or limited deployment methods.

There are three main metrics that are to be evaluated: s the size, complexity of source application, and subjective/objective quality of source application.

In addition, there are several questions that should be considered while evaluating the portability of an application: What is the application's original implementation language? Is it possible to modify the new ported version of the application? What other applications does the source application communicate with or utilize?

We assign a numerical value to each metric ranging from zero being the worst and ten for being the best. Currently each metric is evaluated subjectively, but as the framework is developed it will have more objective means of determining quality. The main objective is to determine whether it is worthwhile or epossible to move the application to a mobile device.

The size of an application can be measured in a number of ways. In our paper, we sacrifice accuracy for simplicity at this time by simply measuring the physical SLOC (Source Lines of Code). Although it is risky to simply determine the size of the program by source code, these metrics are used to quickly determine if portability is worth the effort. Although concrete numbers for SLOC have yet to be determined in our approach, a general example of a ten rating would be a system which numbers of 1,000,000 SLOC. A rating of five could be considered 50,000 SLOC and a rating of zero would be for programs under 1,000 SLOC.

Complexity is the measure of the quality of different traits such as cohesion, coupling, and the level of inheritance. Rather than deal with low level complexity such as execution complexity, the meaning of complexity is denoted at higher level abstractions in UML diagrams. An optimistic view would be to hope that developers would keep metrics on cohesion, coupling, inheritance and other object-oriented metrics [18] [19]. By combining these metrics, the developer could determine the complexity of the application as a whole.

Quality is another metric which depends on the developer's discretion. The factors which a developer should evaluate include the following: readability, ease of maintenance, resource consumption, and number of faults. In addition factors like reliability, correctness, and documentation should also be considered. A program that is coded and documented poorly, contains faults, and uses high resources would score poorly while a higher 'quality' application would have good documentation, be properly coded, and efficiently use resources.

B. Transition to PIMs / PSMs

In this paper, we propose the use a combination of round-trip engineering, MDD concepts, and platform independent/specific models from MDA to effectively port applications onto mobile devices. Although this approach attempts to leverage automated model transformations and code generation to ease the task of transitioning to mobile environments, there will always be a degree of required human interaction at various points to accommodate the deficiencies in automated tools. Depending on the source and target platforms, the range of difficulty can be varied because of the uncertainty related to the incompatibilities between language constructs and platforms.

If there are no models available of the application's design, the first step of the transition is to convert source code into a platform specific model using reverse engineering. If a model exists already, it will be used to create a PIM version if the application is complex enough. There are several tools offered that provide the capability for reverse engineering source code into UML models like Rational ROSE Enterprise, Rational Software Architect and Eclipse [15]. Further research is needed to determine which tool would be best suited towards achieving this goal. One example of reverse engineering source code into models is shown in the mini case study. The objective is to obtain a model version of source code that is as consistent as possible with the source code. After a PSM has been obtained, the next step is to develop either develop a PIM model if the application is going to be deployed to multiple platforms or by converting the existing PSM to a PSM for the target platform directly.

C. Deployment to Target Platform

The last proposed step is possibly the most complex due to additional constraints on mobile devices at both hardware and software levels. It's not as simple as converting an existing application into a new language but may also involve issues such as refactoring classes and splitting the application into a server and client context in order to obtain a working deployment. Other problems that may be encountered include the lack of support from tools to automatically generate code for target platforms, inconsistencies between the platform independent model and the platform specific model, or inadequate deployment models.

Depending on the type of model obtained in the previous step, the last steps involve either converting a PIM to a PSM or using the PSM from the initial conversion of source code to model form. The final portion involves modifying the platform specific models so that they fit into the requirements of the target application. Depending on how large and complex program is, some models may not require modification to run on a new device. Further research is required to determine what sort of issues can be encountered at the final portion before code transformation.

The final step is to take the PSM models that are ready for deployment and forward engineer them into the target platform's source code. Once the source code has been created, further work is required to ensure that the new application works as closely possible to the original version. This final step can be limited to the tools ability to forward engineer to the languages the platform supports. If this is the case, the developer will have to manually create the source code from the models. This is unwanted and research will have to be done to deal with conversions to newer languages as they come into existence.

IV. MINI CASE STUDY – THE IPHONE

The preliminary framework proposed above assumes that the source and target platforms allow for some level of developer freedom in choosing the implementation method on the target platform. The problem we address in this section deals with the result of placing additional unique constraints on an already constrained device.

To evaluate this problem, the iPhone was decided upon due to the unique restrictions placed on developers who wish to develop applications for the platform. The iPhone has the traditional resource constraints found on most mobile devices as well severe limitations on how applications can be deployed onto the phone.

A. Limitations on the iPhone

Because the iPhone is a mobile device, it has hardware limitations that prevent developers from freely doing what they wish to do in JavaScript or Ajax. These limitations include a 10 MB limit on all content downloaded to the phone including JavaScript, HTML, CSS, and images and a five-second execution limit on all JavaScript functions from each top-level entry point [14]. Because of the limited memory, users are encouraged to limit their use of JavaScript to a minimum so that the performance is not adversely affected. This means that it is not possible to create or port large applications on the iPhone.

In addition, the Safari browser supports cookies and user-initiated window calls but does not support Flash, Java, plug-ins, or mouse-over events in its current state on the iPhone [14]. This means that are developers are limited in how they can design applications for the iPhone.

B. Implications on Software Migration

As expected, the more restrictions are placed on how a certain device operates, the harder it is to create or port software applications onto that device. The iPhone is by no means an exception to this. In the interest of simplicity and understanding, the applications used are not true pervasive applications and are only meant to evaluate more basic problems developers could run into with applications on restricted devices in general.

In order to test the basic process of the proposed framework, we chose a simple application to port to the iPhone. Due to time constraints, the application source application was a simplified version of a game called Asteroids. In this version by D. Zimardi, the game is a Java applet which tracks the user's mouse movement to aim and mouse clicks to fire bullets to destroy asteroids. Unlike the original Asteroids, the ship is stationary at all times and there are five stages of increasing difficulty. The keyboard is used at certain points to start a new level or reset the game after running out of lives.

As was stated before, the original game was coded in Java and run through a Java applet. The iPhone does not support Java applets through its Safari browser as well as some of the event handlers used by the original version of the game. Because the application is written in Java, it is ready to be reverse engineered into UML class diagrams. Rational ROSE Enterprise was used to reverse engineer the source code into models.

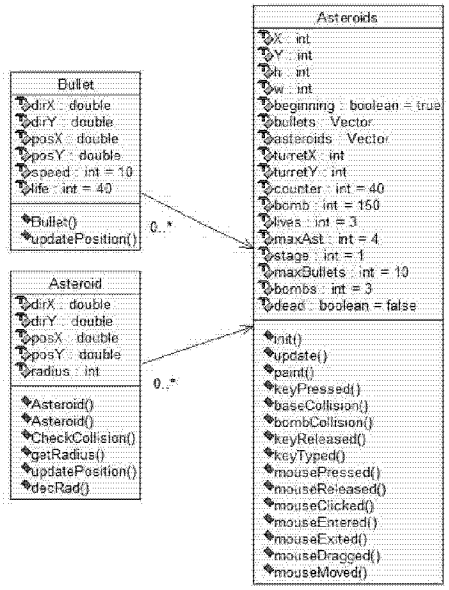

Fig. 1: Asteroids Class Diagram

The three main classes are: the Asteroids class which handles the majority of the game logic and graphic generation, the Asteroid class which defines a single moving asteroid, and the Bullet class which the player uses to destroy asteroids. The Asteroids class also has several event handlers for the mouse and keyboard which are used to control the game. Images, graphics, and sound are created and loaded through Java classes. The classes are show in Figure 1.

For portability evaluation portability, the application receives a score of 1 on size and complexity since it is less than 1,000 LOC and a score of 7 on quality. These evaluations are still subjective and need further work to integrate metric tools to evaluate concrete statistics.

Because the application is simple, the model obtained from the source implementation was modeled directly to a JavaScript implementation. This presented

some unique problems later on due to the limitations of JavaScript and Rational ROSE, which will be discussed in the following sections.

C. Transitional Problems.

The target implementation model could not be obtained automatically inside of the Rational ROSE tool. It provides no support for automatic model transformations to different implementation languages. Thus, the PSM for JavaScript had to be created manually. Worse yet, Rational ROSE does not support concrete support for JavaScript either which only compounded the difficulty of this step. The resulting PSM is show in Figure 2 below:

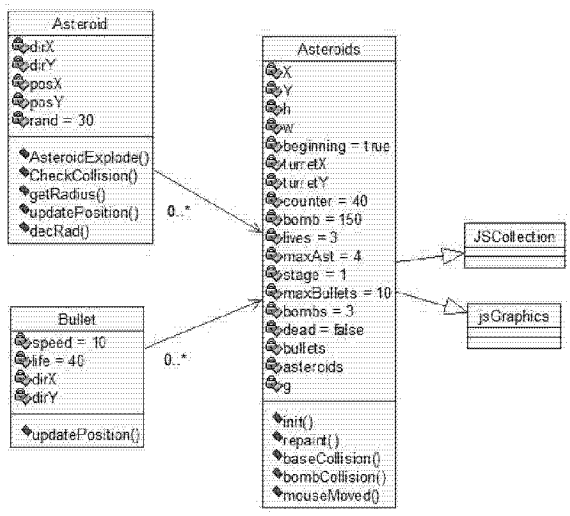

Fig. 2: Asteroids JavaScript PSM

This presented some problems because classes are comprised differently in JavaScript; classes in JavaScript are initially declared as functions with member functions declared as prototype members. Rather than simply reconstruct the new models in a similar fashion, every variable does not have a specific data type determined at compile time. Functions do not assign explicit data types to their classes unlike Java. This means that the effectiveness of modeling for such a weak-typed language is reduced.

Also, JavaScript has no support for function overloading which meant that a second constructor for the Asteroid class had to be renamed as a public function of the class. This was only a minor nuisance but is typical of the problems of converting an application written in a strong-typed language to a weak-typed language. It is questionable if MDD tools could even account for such minor differences in languages or if it requires the developer to be aware of such nuances.

It may be possible that if Rational ROSE had supported concrete modeling for JavaScript that these problems would be a non-issue. However, if a developer had to manually redesign a much larger and/or complex application into a series of JavaScript classes, the problem would be much more serious. Unfortunately, this is a non-option for anyone developing any type of application for the iPhone. It may not be possible to create an advanced pervasive application for the iPhone if basic Java applet games cannot be easily recreated into JavaScript.

D. Deployment Problems

In addition to providing no support for MDD, Rational ROSE Enterprise does not support forward engineering (code generation) for JavaScript. This meant that implementation had to be done manually with the use of the target PSM. This is a common theme with MDD because of the limited availability of actual tool support for the methodology. Currently, it seems that MDD is more-so a concept rather than a viable physical method to use.

Furthermore, JavaScript does not include language support for Vectors or advanced graphic generation. To correct this, two external JavaScript files had to be included: one for a JavaScript implementation for Vectors and other for drawing basic geometrical shapes and image generation. Fortunately, these implementations nearly mirrored the implementations in Java. Other changes involved making variables compatible with JavaScript classes and removing incompatible mouse events.

One major problem encountered that was not expected was a failing in the setInterval () function in JavaScript. It was needed to consistently refresh the web page DIV element to give the appearance of movement and is a function provided by JavaScript. In the first iteration of the model, the event handler for a mouse click called the setInterval () function to repaint the screen every 100 milliseconds. The repainting of the screen is included with the Java language. In JavaScript, we were forced to use an open-source implementation to draw geometric shapes and images into the DIV element.

Either way, due to deliberate design or unintended fault, the setInterval () function provided by JavaScript cannot access public members of classes. This forced us to redesign the Asteroids class. In order for the application to work as the original had, the main Asteroids class had to be deconstructed to plain JavaScript global variables and functions. From there,

the setInterval () function could be called from the HTML page from which the application runs.

The difficulties in translating between languages can be seen. Rather than being allowed to use a more flexible and robust language such as Java, Apple decided to limit developers and users to the more 'secure' Safari browser.

V. CONCLUSION

Pervasive systems are still an emerging area of research in mobile computing. The unique challenges presented to pervasive applications by these systems require an adaptive approach that simplifies the inherent complexity in pervasive environments. The preliminary framework presented in this paper proposes the leveraging of existing techniques in software engineering to enable the creation and transitioning of software to mobile devices. As demonstrated by the test implementation, there are sometimes unexpected consequences in the different implementation languages alone. Factor in the complexity of pervasive applications, additional resource constraints, and problems in automated model transformations and the problem becomes much more difficult to address.

As other work progresses in the area of pervasive computing, the field will begin to resemble more closely the traditional arena of software development on desktop machines. Just as software from twenty years ago only provided basic functionality, the pervasive applications of today are in an immature state. The field is continually growing as researchers work to address areas like context-sensitivity, device and network heterogeneity, and power issues. As advances are made in these areas, further work can be done in developing a complete and robust framework.

In this paper we presented a framework for converting applications to mobile platforms. The basic process involves converting the source application from code format to model form. From there, the platform specific source model is either redesigned into a PIM which describes the business logic of the application. If the application is simple, the source model is converted directly into a target platform's language. If a PIM is obtained, it is transformed into unique platform specific models for each platform. From that point, forward engineering is used to generate code from the PSMs. In this paper, we applied the basics of the framework to an application that was to be converted to run on the iPhone.

As expected, the process ran into various problems. There is limited MDD tool support available for the various steps of this process. One tool may provide functionality for certain steps but not every single step of the framework. This resulted in a good deal of manual intervention to supplement the portions that are not automated by tools yet like model transformation and code generation.

The initial experience with developing for the iPhone suggests that it may not be viable to create pervasive applications for the platform. The limitations of the language are simply too great to allow for developers to easily create any type of advanced application for it. Apple has promised that in the future a SDK will become available and is currently slated for a release in February 2008 but no details have been given as of yet. In its current state it is simply too uncertain and difficult to create advanced applications on the iPhone.

In the future, further work will be done to integrate specific tools into the process and enable developers to have multiple options in porting software. Additional problems may present themselves in the future as issues such as context-awareness appear when pervasive applications are ported using this framework.

REFERENCES

[1] Liu, R., Chen, F., Yang, H., Chu, W.C., Yu-Bin L.; "Agent-based Web services evolution for pervasive computing." *In Proceedings of the 11th Asia-Pacific Software Engineering Conference 2004* (Busan, South Korea, Nov. 30th – Dec. 3rd 2004). pp. 726-731.

[2] Gazzotti, M.; Mamei, M.; Zambonelli, F.; "A programmable event-based-middleware for pervasive mobile agent organizations." *In Proceedings of Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing* (Feb. 5-7, 2003). pp. 517-524.

[3] Xiaoli, Z., Rong, L., Tong, W.; "Adaptive Middleware for Uniform Access to Legacy Systems for Mobile Clients." *In Proceedings of the 1st International Symposium on Pervasive Computing and Applications (Urumqi, August 3rd – 5th, 2006)*. pp. 46-50.

[4] Weis, T.; Handte, M.; Knoll, M.; Becker, C.; "Customizable pervasive applications." *In Proceedings of Fourth Annual IEEE International Conference on Pervasive Computing and Communications.* (PerCom Mar. 13-17, 2006.) pp. 6-14.

[5] Reilly, D.; Dearman, D.; Welsman-Dinelle, M.; Inkpen, K.; "Evaluating early prototypes in context: trade-offs, challenges, and successes." Pervasive Computing, IEEE , vol.4, no.4, pp. 42-50, Oct.-Dec. 2005

[6] Weis, T.; Knoll, M.; Ulbrich, A.; Muhl, G.; Brandle, A.; "Rapid Prototyping for Pervasive Applications." Pervasive Computing, IEEE , vol.6, no.2, pp.76-84, April-June 2007

[7] Pham, Huy N; Mahmoud, Q.; Ferworn, A., Sadeghian, A.; "Applying Model-Driven Development to Pervasive System Engineering." *First International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments.* (SEPCASE 2007: May 20-26 2007). pp.7-7.

[8] Mascolo, L. Capra, W. Emmerich, "An XML based Middleware for Peer-to-Peer Computing." *In Proc. of the International Conference of Peer-to- Peer Computing 2001*. IEEE CS Press.

[9] Picco G., Murphy A., Roman G.; "LIME: A Middleware for Logical and Physical Mobility." 13th International Conference on Distributed Computing Systems, Linda Meets Mobility", July 2001.

[10] G. Cugola, A. Fuggetta, E. De Nitto.; "The JEDI Event-based Infrastructure", IEEE Transactions on Software Engineering, 27(8), August 2001.

[11] G. Kleppe, J. B. Warmer, W. Bast, and A. Watson, MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Professional, 2003.

[12] Ferworn, A., Sadeghian, K., Barnum, H., Rhanama, H., Pham, C., Erickson, D., Dell'Agnese L.; "Urban Search and Rescue with Canine Augmentation Technology," *In Proceedings of IEEE System of Systems Engineering* (SoSE 2006 Los Angeles, CA).

[13] Becker, C., Handte, M., Schiele, G., Rothermel, K.; "PCOM – A Component System for Pervasive Computing." *IEEE Intl' Conf' on Pervasive Computing and Communication.* (March 2004). pp. 67-77

[14] Apple Computer Inc. "iPhone Dev Center - Apple Developer Connection", Available at: http://developer.apple.com/iphone/devcenter/designingcontent.html

[15] IBM Corporation, "IBM – Rational ROSE – Family Overview" Available at: http:/www.ibm.com/software/awdtools/developer/rose/index.html

[16] M. Ilyas, I. Mahgoub, Mobile Computing Handbook. Auerbach Publications, 2004.

[17] Alan Brown: "An Introduction to Model-Driven Architecture (MDA)." IBM. Retr. at March 7, 2008 at http://www-128.ibm.com/developerworks/rational/library/apr05/brown

[18] Meyers, T.M.; Binkley, D.; "Slice-based cohesion metrics and software intervention." *Proceedings of 11th Working Conference on Reverse Engineering.* (Nov. 8-12 2004). pp. 256-265.

[19] Liu, Dapeng; Xu, Shaochun, "New Quality Metrics for Object-Oriented Programs," *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing.* (SNPD 2007 Jul. 30 - Aug. 1, 2007). vol.3, pp.870-875.